6. Hash

Program Name: Hash.java   Input File: hash.dat

Hash tables are great for making a lookup table with "amortized constant time complexity lookup". It's okay if those words make no sense; they aren't important. Hash tables rely on hashing algorithms that assign unique indices to their input keys. The problem is that sometimes the algorithm will assign the same index for two different input keys. When this happens, a collision occurs. To solve a collision, the hash table might put one of the keys in an index nearby, but this does not always work either. You will use the following hash algorithm:

Given a String of characters [C0][C1][C2]…[C(N-1)], the index of this string (key) is calculated by summing each character's value multiplied by 2 to the power of its index in the String, and then taking a modulo the size of the table M. Each character's value is its distance from 'a'. In equation form,

$$index = \left(\sum_{i=0}^{i=N-1} (Ci - 'a') * 2^i \right) \% M$$

If this index is already taken, the key can be placed in the location directly above it, or directly below it (Pretend that the beginning and end of the table are connected). If both of these are taken, the collision cannot be resolved. Keys cannot be rearranged after being put in and must be added in the order given. When a collision occurs, consider both if the key is placed either one above or one below. You will test both placements for any collision and return the minimum number of unresolved collisions that will occur in the best case scenario.

For example, in a size 3 hash table, with three elements, 'a', 'b', and 'c', the 'a' hashes to $(0*2^0)\%3$, which equals 0, so the 'a' goes into position 0 in the hash table. Likewise, 'b' and 'c' hash to 1 and 2 and go into those table positions, with no unresolved collisions.

In another example, with 4 elements and a size 5 table, the element "afd" would hash to the following:

$$((0*2^0) + (5*2^1) + (3*2^2))\%5 = (0 + 10 + 12)\%5 = 22\%5 = 2,$$

placing it into position 2 of the table. An element "ba" would hash to position 1, and "ccs" would go into position 3. A fourth element "eee" would hash to the same location containing "ccs", position 3. But since the position just above (location 4) is available, "eee" goes there, and all is well with the world.

However, if the fourth element was "dee", it would hash to the value 2, which already contains "afd". To resolve this collision, you would look directly above and below, but those two positions are also taken, and therefore "dee" has nowhere to go, and we have 1 unresolved collision.

Input

The first line will contain T, the number of unique test cases. Each test case will start with two integers M and N, where M is the size of the hash table, and N is the number of strings to put into the hash table. This is followed by N lines, each with an input key string.

Constraints

```
0 < T < 30
0 < M < 20
0 < N < 20
```

Output

For each test case, print the number of unresolved collisions that will occur in the best case.

Example Input File

```
3
3 3
a
b
c
5 4
afd
ba
ccs
eee
5 4
afd
ba
ccs
dee
```

Example Output to Screen

```
0
0
1
```